Week 11

COMP3231 Operating Systems

2005 S2

- ➜ Today: Real-time systems, wrap up
- ➜ Today: File Systems (Chapter 6, Tanenbaum)
  - User's view
  - File System Implementation
- ➜ Tutorial this week: device driver
- ➜ Week 12:
  - Assignment 2 Solution (fork)
  - Case Studies
- ➜ Week 13: Operating System Design
- ➜ Week 14: Overview

LINUX 2.4 SCHEDULING — SOFT REAL-TIME SUPPORT

- ➜ User assigns static priority to real time processes (1-99), never changed by scheduler
- ➜ Conventional processes have dynamic priority, always lower than real time processes
  - sum of base priority and
  - number of clock ticks left of quantum for current epoch
- ➜ Scheduling classes
  - SCHED_FIFO: First-in-first-out real-time threads
  - SCHED_RR: Round-robin real-time threads
  - SCHED_OTHER: Other, non-real-time threads
- ➜ Within each class multiple priorities may be used
- ➜ Deadlines cannot be specified, no guarantees given
- ➜ Due to non-preemptive kernel, latency can be too high for real-time systems

LINUX 2.4 SCHEDULING — SOFT REAL-TIME SUPPORT

- ➜ Virtual Memory:
  - no VM for real-time apps
  - `mlock()` and `mlockall()` to switch off paging (which other applications might need to do this?)
- ➜ Timer: resolution: 10ms, too coarse grained for real-time apps

Linux scheduling:

A	**minimum**
B	**middle**
C	**middle**
D	**maximum**

(a) Relative thread priorities

D ⟶ B ⟶ C ⟶ A ⟶

(b) Flow with FIFO scheduling

D ⟶ B ⟶ C ⟶ B ⟶ C ⟶ A ⟶

(c) Flow with RR scheduling

Slide 5

IMPROVEMENTS IN 2.6 KERNEL

- Kernel Preemption
  - kernel code laced with preemption points
  - calling process can block and thereby yield CPU to higher-priority process
- Kernel can be built without VM
- Improved scheduler
- Timer resolution: 1ms

Slide 6

SCHEDULING IN 2.4 AND 2.6: COMPARISON

2.4:

- CPU time divided into epochs
- Each process has a (poss. different) time quantum it is allowed to run in every epoch
- Epoch ends when all runnable processes have exhausted their quantum
- Time quantum for each process recomputed after every epoch
- To find the next process which should be scheduled, the complete ready-queue has to be scanned
- SMP: only single ready-queue
- $\mathcal{O}(n)$ algorithm: overhead grows linearly with number of processes
- Ready queue access bottle neck for SMP

Slide 7

2.6:

- Queue for each priority
- Thread can be in active (quantum not yet expired) or expired (quantum already used up) queue.
- Priority is re-calculated after quantum is expired
- Interactive processes inserted back into active-queue
- SMP: One set of queues per processor, idle processors steal work from other processors
- $\mathcal{O}(1)$ algorithm: time required for scheduling decision does not depend on number of processes
- Ready queue access not a bottle neck for SMP
- Better locality

Slide 8

HARD REAL TIME OS

We look at examples of two types of systems:

- hard real-time variants of general purpose OSs
  - try to alleviate shortcomings of OS with respect to real time apps
- configurable hard real time systems
  - system designed as real time OS from the start

Slide 9

RTLinux

- ➜ abstract machine layer between actual hardware and Linux kernel
- ➜ takes control of
  - hardware interrupts
  - timer hardware
  - interrupt disable mechanism
- ➜ real time scheduler runs with no interference fron Linux kernel
- ➜ programmer must utilise RTLinux API for real time applications

Slide 10

QNX

- ➜ Microkernel based architecture
- ➜ POSIX standard API
- ➜ Modular — can be costumised for very small size (eg, embedded systems) or large systems
- ➜ Memory protection for user applications and os components

Slide 11

Scheduling:

- ➜ FIFO scheduling
- ➜ Round-robin
- ➜ Adaptive scheduling
  - thread consumes its timeslice, its priority is reduced by one
  - thread blocks, it immediately comes back to its base priority
- ➜ POSIX sporadic scheduling

Slide 12

Kernel Services:

- ➜ Thread services: provides the POSIX thread creation primitives.
- ➜ Signal services: provides the POSIX signal primitives.
- ➜ Message passing services: handles the routing of all messages between all threads through the whole system.
- ➜ Synchronization services: provides the POSIX thread synchronization primitives.
- ➜ Scheduling services: schedules threads using the various POSIX realtime scheduling algorithms.
- ➜ Timers services: provides the set of POSIX timer.

Slide 13

Process Manager:

The process manager is capable of creating multiple POSIX processes (each of which may contain multiples POSIX threads).

Its main areas of responsability include:

- ➜ Process management: manages process creation, destruction, and process attributes such us user ID and group ID.
- ➜ Memory management: manages memory protection, shared libraries, and POSIX shared memory primitives.
- ➜ Pathname management: manages the pathname space (mountpoints).

Slide 14

WINDOWS CE 5.0

Componentised OS designed for embedded systems with hard real-time support

- ➜ handles nested interrupts
- ➜ handles priority inversion based on priority inheritance

Offers

- ➜ guaranteed upper bound on high priority thread scheduling
- ➜ guaranteed upper bound on delay for interrupt service routines

Slide 15

FILE SYSTEMS

Long-term information storage:

① Must support storage of lager amount of data

② Information must survive termination of process creating the information

③ Multiple processes must be able to access information concurrently

Slide 16

Information is stored in files

- ➜ on disk or other external media
- ➜ processes can read, write, and create new files
- ➜ a file should only disappear when explicitely removed by owner

The OS component which manages files is called the file system

Concrete file system determines:

- ➜ structure
- ➜ implementation
- ➜ usage
- ➜ protection

Slide 17

Why is the file system part of the operating system?

- ➜ Manages trusted, shared resource
- ➜ Provides abstraction layer:
  - hides low-level disk organisation
  - presents it to the user as a collection or stream of records

Included set of tools outside of kernel:

- ➜ formatting
- ➜ recovery
- ➜ defragmentation
- ➜ back up

Slide 18

OBJECTIVES

- ➜ Provide convenient user interface
- ➜ Provide uniform I/O support for a variety of storage devices
- ➜ Optimise performance
- ➜ Provide security and safety

Slide 19

FILE NAMING

File system must provide a convenient naming scheme:

- ➜ textual names
- ➜ namespace may be restricted
  - exclude certain characters
  - limited length
  - only certain format (DOS 8+3)
- ➜ names may obey conventions
  - interpreted by tools (UNIX)
  - interpreted by operating system (Windows)

Slide 20

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

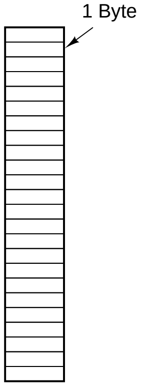
Slide 21

FILE STRUCTURE

File as Byte Sequence:

- ➜ operating system does not know about the contents of the file
- ➜ meaning imposed by user-level program
- ➜ approach used by Windows, Unix
- ➜ provides maximum flexibility

Slide 22



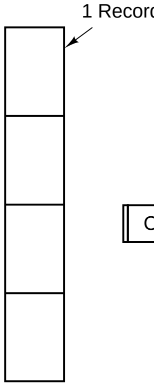
(a)

Slide 23

File as Collection of Fixed-length Records:

- ➜ each record has internal structure
- ➜ read and write operations record oriented
- ➜ was used in many mainframe operating system
- ➜ not used in any current general purpose operating system

Slide 24

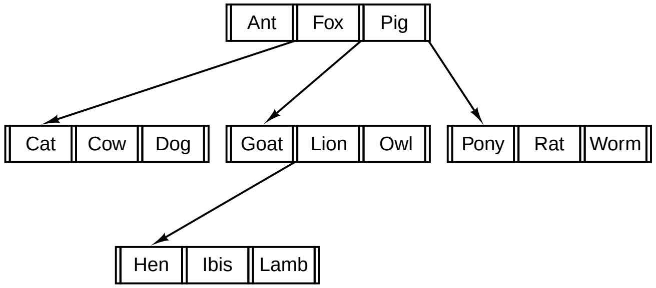


(b)

File as Tree of Records:

- ➜ not necessarily of the same size
- ➜ access record through key
- ➜ os, not user level program places new records
- ➜ used for large scale data processing on some main frame systems

Record



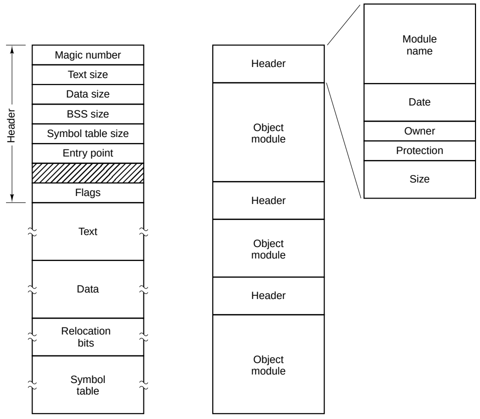
(c)

FILE TYPES

- ➜ Regular files
  - ASCII text files
  - binary files
- ➜ Directories
- ➜ Device files
  - character devices (stream of bytes)
  - block devices

All system recognize their own executable format (often identified by magic number)

FILE STRUCTURE



(a) (b)

Slide 29

FILE ACCESS

- ➜ Sequential Access
  - read all data from the beginning
  - can’t move back, only rewind
  - convenient for magnetic tape
- ➜ Random Access
  - read data in any order
  - essential for applications which use large files (data base etc)
  - start position can be either set by each call to read, or set by special seek instruction

Slide 30

FILE ATTRIBUTES

- ➜ in addition to name and data, file attributes are stored
- ➜ set of attributes associated with a file depends on OS
- ➜ categories:
  - protection
  - time stamps
  - type of file

Slide 31

FILE ATTRIBUTES

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Slide 32

FILE OPERATIONS

- ➜ Create / Delete
- ➜ Open / Close
- ➜ Read /Write
- ➜ Seek
- ➜ Get / Set attributes
- ➜ Append
- ➜ Rename

Slide 33

```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                    /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);         /* ANSI prototype */

#define BUF_SIZE 4096                     /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                  /* protection bits for output file */

int main(int argc, char *argv[])
{
    int in_fd, out_fd, rd_count, wt_count;
    char buffer[BUF_SIZE];

    if (argc != 3) exit(1);               /* syntax error if argc is not 3 */

    /* Open the input file and create the output file */
    in_fd = open(argv[1], O_RDONLY);      /* open the source file */
    if (in_fd < 0) exit(2);               /* if it cannot be opened, exit */
    out_fd = creat(argv[2], OUTPUT_MODE); /* create the destination file */
    if (out_fd < 0) exit(3);              /* if it cannot be created, exit */
```

Slide 34

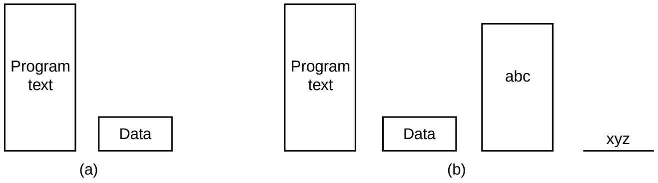
```
    if (argc != 3) exit(1);               /* syntax error if argc is not 3 */

    /* Open the input file and create the output file */
    in_fd = open(argv[1], O_RDONLY);      /* open the source file */
    if (in_fd < 0) exit(2);               /* if it cannot be opened, exit */
    out_fd = creat(argv[2], OUTPUT_MODE); /* create the destination file */
    if (out_fd < 0) exit(3);              /* if it cannot be created, exit */

    /* Copy loop */
    while (TRUE) {
        rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
    if (rd_count <= 0) break;             /* if end of file or error, exit loop */
        wt_count = write(out_fd, buffer, rd_count); /* write data */
        if (wt_count <= 0) exit(4);       /* wt_count <= 0 is an error */
    }

    /* Close the files */
    close(in_fd);
    close(out_fd);
    if (rd_count == 0)                    /* no error on last read */
        exit(0);
    else
        exit(5);                          /* error on last read */
}
```

Slide 35

MEMORY-MAPPED FILES

This style of accessing files is inconvenient

- ➜ `unmap`
- ➜ `map`
- ➜ virtual address region backed by file
- ➜ easy to realise if system supports segmentation



Slide 36

Potential Problems:

- ➜ consistency, if multiple processes access file
- ➜ file may be too large to fit in address space

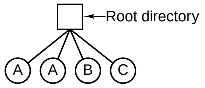
Slide 37

DIRECTORIES

- ➜ Contain information about files
  - attributes
  - location
  - ownership
- ➜ directory itself is file owned by os
- ➜ provides mapping between filenames and actual files

Slide 38

SINGLE-LEVEL DIRECTORY SYSTEM

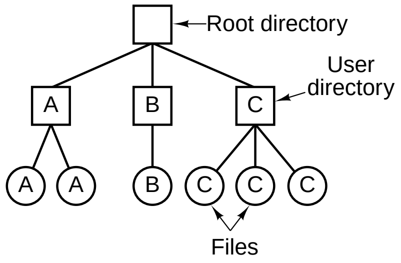


- ➜ used on early personal computers, first supercomputer (CDC6600)
- ➜ no filename can be used twice
- ➜ problematic for multiuser systems
- ➜ no help for organising files
- ➜ sufficient for small embedded systems etc

Slide 39

TWO-LEVEL DIRECTORY

- ➜ master directory contains one entry per user (access control information)
- ➜ user directory simple list of files owned by the user
- ➜ still no support for file organisation
- ➜ need for system directory containing shared executables

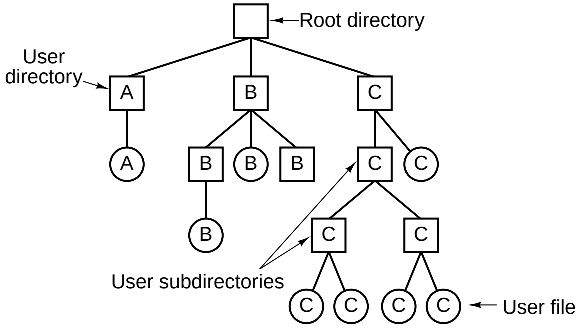


Slide 40

HIERARCHICAL DIRECTORY SYSTEMS

- ➜ master directory with user directories underneath
- ➜ each user directory may have subdirectories and/or files as entries
- ➜ files can be located by following a path from the root (or master) directory down (absolute path name)
- ➜ files with the same name possible, as long as the path name differs

Slide 41



Slide 42

WORKING DIRECTORY

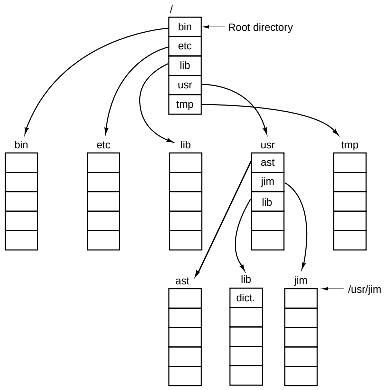
- ➜ the absolute pathname is in general quite long: too tedious to work with
- ➜ introduce the concept of a working directory
  - files can be referenced relative to working directory
  - each process has its own working directory
- ➜ Example: if current working directory `/home/keller`, then `.profile` references the same files as `/home/keller/.profile`

Slide 43

PATH NAMES

- ➜ different syntax in different os's
  - Windows: `\usr\ast\mailbox`
  - Unix: `/usr/ast/mailbox`
  - Windows: `>usr>ast>mailbox`
- ➜ in most hierachical directory systems two special entries:
  - current directory: `.` in Unix
  - parent directory: `..` in Unix

Slide 44



Slide 45

DIRECTORY OPERATIONS

Contents of directory files may not be manipulated by user directly

Unix directory operations:

- create/delete
- open/close
- read directory
- link/unlink

Slide 46

FILE SHARING

- Multi user systems allow files to be shared among users
- How are the access rights handled?
- How is simultaneous access managed?

Slide 47

ACCESS RIGHTS

- None:
  - user may not know of existence of the file
  - not allowed to read directory which includes file
- Knowledge
  - user can only determine that file exists and who the owner is
- Execution
  - user can load and execute, but cannot copy it
- Reading
  - user can read the file for any purpose, including copying and execution

Slide 48

ACCESS RIGHT

- Appending
  - user can add data at the end of the file, but cannot alter or delete the file's previous content
- Updating
  - user can modify, delete, and add to file's data
- Changing protection
  - user can change access rights granted to other users
- Delete
  - user can delete file

Access Rights

Owner

- ➜ has all rights previously listed
- ➜ May grant rights to others using the following classes of users
  - Specific user
  - User group
  - All for public files

Case Study

UNIX access permissions

```
total 1704
drwxr-x---   2 keller keller   4096 Oct  8 18:34 .
drwxr-x---  15 keller keller   4096 Oct  8 18:33 ..
drwxr-x---   1 keller keller   4096 Oct  8 18:33 backup
-rw-r-----   1 keller keller 423444 Oct  8 18:34 bar.txt
-rw-r-----   1 keller keller  12332 Oct  8 18:34 foo.jpg
```

- ➜ First letter: file type
  - `d`: directory
  - `-`: regular file
- ➜ Three user categories:
  - user
  - group
  - other

UNIX access permissions

```
total 1704
drwxr-x---   2 keller keller   4096 Oct  8 18:34 .
drwxr-x---  15 keller keller   4096 Oct  8 18:33 ..
drwxr-x---   1 keller keller   4096 Oct  8 18:33 backup
-rw-r-----   1 keller keller 423444 Oct  8 18:34 bar.txt
-rw-r-----   1 keller keller  12332 Oct  8 18:34 foo.jpg
```

Three access rights per category

- ➜ read
- ➜ write
- ➜ execute

```
drwxrwxrwx
user   other
    group
```

UNIX access permissions

```
total 1704
drwxr-x---   2 keller keller   4096 Oct  8 18:34 .
drwxr-x---  15 keller keller   4096 Oct  8 18:33 ..
drwxr-x---   1 keller keller   4096 Oct  8 18:33 backup
-rw-r-----   1 keller keller 423444 Oct  8 18:34 bar.txt
-rw-r-----   1 keller keller  12332 Oct  8 18:34 foo.jpg
```

- ➜ execute permission for directory?
  - permissions to access files in the directory
- ➜ to list a directory requires read permission
- ➜ What about `drwxr-x--x`?

Slide 53

UNIX ACCESS PERMISSIONS

- Shortcoming
  - three user categories rather coarse
- Example:
  - Joe owns file `foo.bar`
  - wished to keep file private, not accessible to general public
  - wants Bill to be able to read and write
  - wants Peter to be able to read only

Slide 54

ACCESS CONTROL LISTS

Available in most commercial Unix systems, Windows XP professional, SELinux, Linux 2.6:

- data structure (usually table) containing that specifies access rights of individual users or groups
- different implementations in different OS
- POSIX standard for ACLs

Slide 55

Example: using file ACLs in Linux

- `getfacl`
- `setfacl`

```
urmel keller 1006 (~): getfacl R3000.pdf
# file: R3000.pdf
# owner: keller
# group: keller
user::rw-
group::r--
other::r--
```

Slide 56

```
urmel keller 1007 (~): setfacl -m u:chak:rw- R3000.pdf
urmel keller 1007 (~): getfacl R3000.pdf
# file: R3000
# owner: keller
# group: keller
user::rw-
group::r--
user:chak:rw-
other::r--
```

Slide 57

SIMULTANEUS ACCESS

- ➜ most OSes provide mechanism for users to manage concurrent access to files
  - Example: `lockf`, `flock` system calls
- ➜ user may lock entire file or part of file when it is updated
- ➜ mutual exclusion and deadlock are issues for shared access

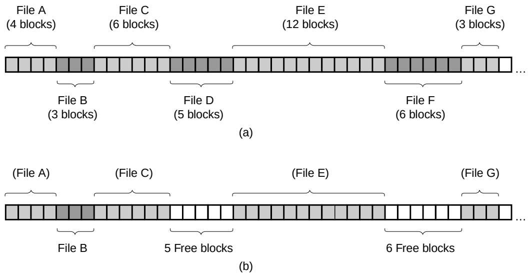
Slide 58

FILE SYSTEM IMPLEMENTATION

How can we map a file to the available space on a hard disk?

Contiguous Allocation:

- ➜ each file stored as contiguous sequence of disk blocks



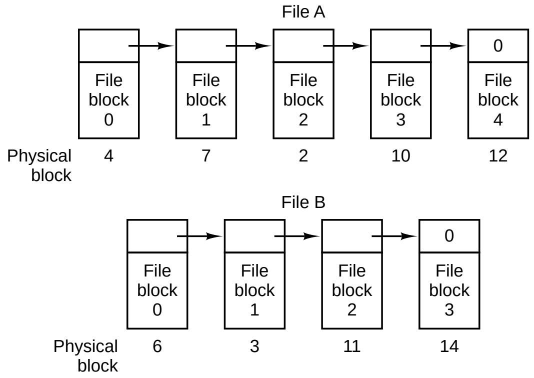
Slide 59

- ✓ simple to implement
  - only necessary to remember start block and no of blocks in file
- ✓ excellen read performance
  - only single seek necessary
- ✗ over time, fragmentation becomes a problem
- ✗ what happens if a file grows in size??
- ✓ good for write-once media (CD-ROM etc)

Slide 60

Linked List Allocation:

Each file is kept as linked list of disk blocks



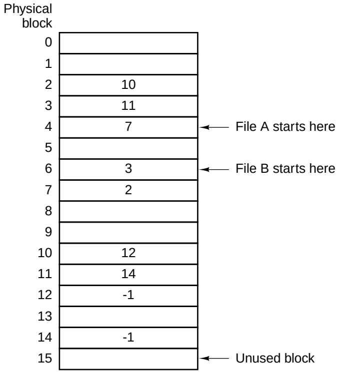
Slide 61

- ✔ still relatively simple to implement
  - only necessary to remember start block
- ✔ (almost) no fragmentation
- ✔ reading file straight forward (but slower than for contiguous allocation)
- ✘ extremely poor random access performance
- ✘ effective block size is not 2^n bytes anymore, as pointer takes up storage

Slide 62

Linked List with Table in Memory:

Using a separate table stored in main memory eliminates both disadvantages:



Physical block		
0		
1		
2	10	
3	11	
4	7	File A starts here
5		
6	3	File B starts here
7	2	
8		
9		
10	12	
11	14	
12	-1	
13		
14	-1	
15		Unused block

Slide 63

- ✔ File Allocation Table FATa
- ✔ entire block available for data
- ✔ random access is much faster and easier
- ✔ directory entry still only needs to store first block of file
- ✘ entire table must be in memory
- ✘ millions of table entries, huge memory consumption

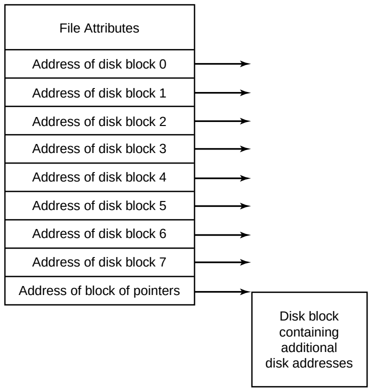
Slide 64

Index nodes (I-nodes):

I-node avoids those disadvantages

- ➜ each file is associated with an i-node
- ➜ i-node has to be in memory only if file is open
- ➜ each i-node contains
  - the attributes of the file
  - disk addresses of the file's blocks
  - straight forward i-node structure only able to store a fixed number of block addresses. What happens if file grows beyond this limit?

Slide 65



File Attributes
Address of disk block 0
Address of disk block 1
Address of disk block 2
Address of disk block 3
Address of disk block 4
Address of disk block 5
Address of disk block 6
Address of disk block 7
Address of block of pointers
Disk block containing additional disk addresses

Slide 66

IMPLEMENTING DIRECTORIES

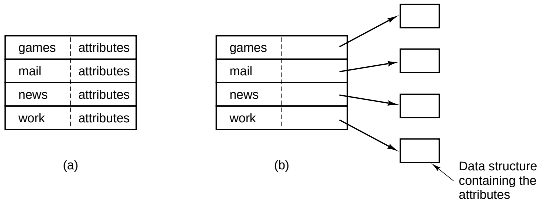
Main function of directory is to map the ASCII name of the file to the information necessary to locate data

- ➜ Contiguous allocation: disk address of file
- ➜ Linked lists: number of first block
- ➜ I-nodes: number of i-node

Attributes:

- ➜ can be stored in the directory itself, or
- ➜ in i-nodes

Slide 67



games	attributes
mail	attributes
news	attributes
work	attributes

(a)

games
mail
news
work

(b)

Data structure containing the attributes

Slide 68

Managing File Names:

- ➜ old OSes often support only short file names:
  - MS-DOS: 8+3 characters
  - Unix, Version 7: 14 characters
- ➜ conceptually easy to increase the limit, but wasteful

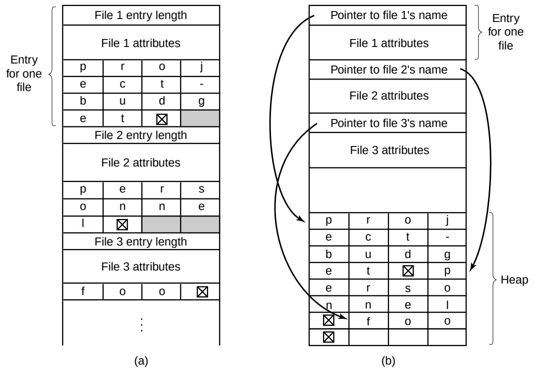
Slide 69

Variable Length File Names:

Two main approaches:

- ➜ In-line storage
- ➜ Heap storage

Slide 70

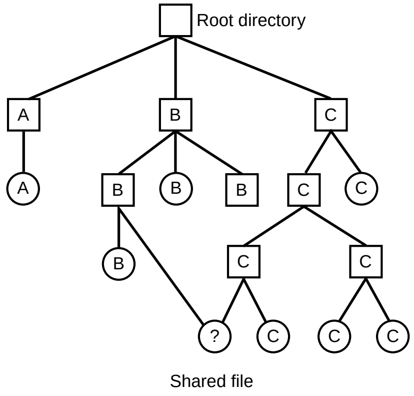


Slide 71

- ➜ In-line storage
  - \- fragmentation
- ➜ Heap storage
  - \- no fragmentation
  - \- no need for names to start at word boundaries

Slide 72

SHARED FILES

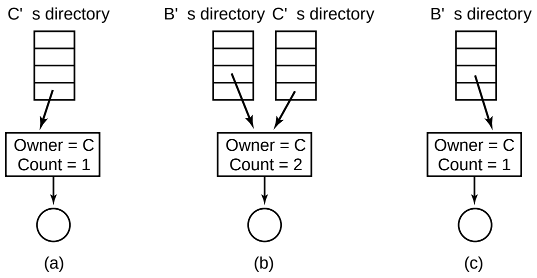


Slide 73

SHARED FILES

- ➜ file tree becomes a directed acyclic graph (DAG)
- ➜ if directory contains disk addresses, copy has to be made
  - what happens if the file size changes?
- ➜ hard link:
  - copy points to the same i-node
  - need to maintain a counter for each file
- ➜ symbolic link:
  - link is new file type
  - Unix: just the file name
  - removing the file can lead to stale links
  - deleting the link has no effect on the file

Slide 74



Slide 75

DISK SPACE MANAGEMENT

We discussed to ways to organise disk memory:

- ➜ allocation of contiguous area on disk
- ➜ split files into blocks

Similar problem as in RAM management (segmentation/paging)

Almost all file systems divide files into fixed equal sized blocks

Slide 76

Optimal Block Size:

What are the trade offs when choosing the block size?

- ➜ too small:
  - files consist of too many blocks
  - overhead
  - extra seeks and rotational delays: reading a file will become slow
- ➜ too big:
  - internal fragmentation
  - wasteful

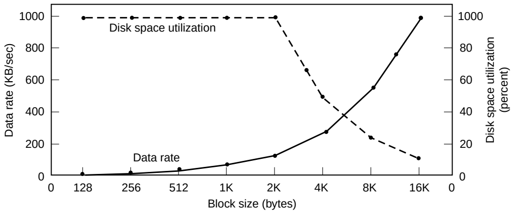
Slide 77

File size statistics (large Unix system, Tanenbaum)

- ➜ mean: 10,845 bytes
- ➜ median: 1680 bytes

Observations on similar type of Windows system lead to comparable results

Disk Utilisation and Data Rate:



Slide 78

FREE BLOCK MANAGEMENT

Two widely used methods

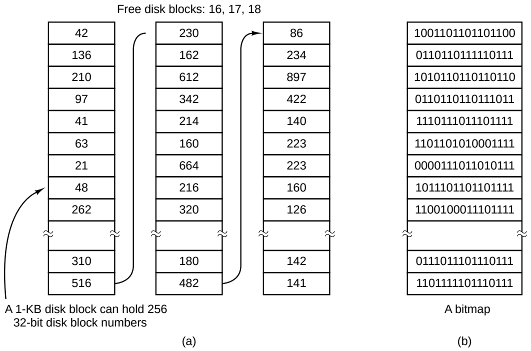
Linked list of Blocks:

- ➜ use a linked list of blocks
- ➜ each block contains disk block numbers of free blocks (number depends on block size)
- ➜ last entry is pointer to next block
- ➜ use free blocks to store the information
- ➜ example: 16GB disk needs 16,794 blocks to hold all numbers
- ➜ only one block needs to be kept in main memory

Bitmap:

- ➜ disk with n blocks requires disk map with n bits
- ➜ 16GB disk needs 2048 blocks to store bitmap

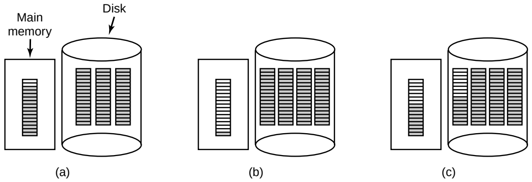
Slide 79



Slide 80

Linked list of Blocks:

- ✘ needs more space than bitmap when disk is empty
- ✔ needs less space when disk is almost full
- ✘ can lead to unnecessary disk I/O



Slide 81

Bitmaps:

- ✗ search through bitmap when few blocks are free
- ✓ easier to allocate contiguous blocks for file